3-1-1999

Very Low-Complexity Digital Filters Based On Computational Redundancy Reduction

Khurram Muhammad
Purdue University School of Electrical and Computer Engineering

Kaushik Roy
Purdue University School of Electrical and Computer Engineering

VERY LOW-COMPLEXITY DIGITAL FILTERS BASED ON COMPUTATIONAL REDUNDANCY REDUCTION

KHURRAM MUHAMMAD
KAUSHIK ROY

TR-ECE 99-5
MARCH 1999

SCHOOL OF ELECTRICAL
AND COMPUTER ENGINEERING
PURDUE UNIVERSITY
WEST LAFAYETTE, INDIANA 47907-1285

Very Low-Complexity Digital Filters Based On Computational Redundancy Reduction[1]

Khurram Muhammad and Kaushik Roy
Email: khurram@ecn.purdue.edu,kaushik@ecn.purdue.edu
School of Electrical and Computer Engineering,
Purdue University, West Lafayette, IN 47907

February 22, 1999.

Abstract

We present computation reduction techniques which can be used to obtain multiplierless implementations of *finite impulse response (FIR.)*digital filters. The ideas presented in this work are also applicable to *infinite impulse response* (IIR) digital filters. The main idea is to remove *computational redundancy* by *reordering* computation. Hence, the frequency response of the desired filter is unaltered. Various approaches are presented which consider *normal, differential* and *hybrid* coefficients. It is shown that the reordering problem can be formulated using a graph in which vertices represent the coefficients and edges represent resources required in a computation involving the coefficient. We present various schemes which reduce filter complexity by specifically targeting computational redundancy inherent in normal filter implementation,~Simple polynomial run time algorithms are presented and their power and potential is demonstrated by presenting results for large (up to 600 tap) filters which show significant reduction in the number of add operations per coefficient. We also consider filter implementations in which *shifted* values of computations can be obtained using simple interconnects without incurring extra computation. We present a, methodology using which such computation can be *re-used* in subsequent computations and show such operations further reduce computational redundancy resulting in extremelly simple filters. It is shown that as low as 0.1 adders per filter coefficient are required to implement the multipliers in such filters. Hence, such filters can be used in very high-speed applications. Alternatively, using voltage scaling, one can significantly reduce the power consumption of such filters for any desired, performance.

'This work was supported in part by DARPA (F33615-95-C-1625), NSF CAREER award (9501869-MIP), Rockwell. AT&T arid Lucent foundation.

I. Introduction

Future mobile radio and portable computing systems are expected to provide increased services, faster data rates and higher processing speeds at reduced power dissipation levels. This provides us with a motivation to explore new approaches in low-complexity design of high-performance *digital signal processing* (DSP) blocks which operate at lower power levels. The *classical* approach [1], [2] in complexity reduction is the use of techniques such as *recursion* (e.g. RLS, FFT algorithms), *multi-rate signal processing* and *low rank approximation*. The last technique is a widely used approach which compromises accuracy by removing less significant computations from a given computational algorithm. Low-complexity design not only improves the speed at which the algorithm can process data, but it also leads to low power design at the highest level of abstraction by reducing energy consuming operations.

In this paper, we explore complexity reduction from the point of view of reducing the number of operations by *reuse* of computation. The new insight provided to the subject of complexity reduction is the reduction of *computational redundancy* which is defined as the *excess computation over the minimum number of bit operations needed for a given sequence of operations.* This approach can be used to compliment existing complexity reduction methods as it reduces the number of energy consuming operations by reusing parts of computation. Although, the idea of computation reuse appears in many forms in typical DSP system implementations, this paper develops and formalizes this approach to the case of FIR filtering in order to illustrate its potential. The proposed techniques are also applicable to direct form IIR filter implementations.

Many previous work have been reported on complexity reduction of FIR filters [3], [4], [5], [6], [7], [8], [9] which consider simplified parallel implementations of FIR filters for *signed powers-of-two* (SPT) implementations. The proposed methods start from a known optimal filter solution and search for *quantizations* in the vicinity of the solution which give lower implenientation cost. Search algorithms have been proposed to obtain "good" solutions to the coefficient quantization problem for *canonical sign-digit* (CSD) number representation [3], [4], [5], [6], [7], [8], [9]. The published results exist for filters of small lengths and 'indicate that more than 2 adders are required per coefficient using such search. One disadvantage of these methods is that these compromise the frequency response of the filter during their search for a lower cost implementation. This deviation in frequency response may not be tolerable in wireless communications where such deviations can increase *multi-user interference.* Further, methods which yield optimal solutions

are computationally expensive for large filters.

Low power FIR filter realizations have also been extensively studied in recent years [10], [11], [12], [13]. The basic techniques used in power reduction constitute *architectural transformations, sub-structure sharing, quantizations,* and *computation reordering.* The idea of *computation reordering* was explored in [11] in the *differential coeficient method* (DCM) which reduces energy consumption by reducing the dynamic range of computation. The main idea is to compute the filter output using coefficient differences instead of their original values. In FIR filters, the dynamic range of difl'erential coefficients is smaller, hence, the width of multipliers can be reduced.

In this paper, we present various approaches to reduce redundant computation in digital filtering. In particular, we address methods which exploit computation reuse in the example case of FIR filtering. The main idea is to find an ordering of coefficients which minimizes the number of adders required in the filter implementation using graph theoretic approaches. We propose *differential coeficients multiplierless implementation* (DCMI) scheme which is shown to significantly reduce filter complexity. We also present optimal solution to the DCMI problem which we referred to as the *optimal differential coeficients multiplierless implementation* (ODCMI) scheme. In general, less than 2 add operations per coefficient are required in 16-bit implementation of unscaled coefficients (3 for *maximally scaled* coefficients) using these approaches. We also present a methodology which further reduces computational redundancy by re-using *shifted* values of a computation in the evaluation of a subsequent computation. The shift operation can be obtained free of computational cost by using interconnect wiring. We refer to this technique as the *minimally redundant parallel filters* (MRPF) technique. Results indicate that, in general, less than 1 adder per coefficient is required for 16-bit maximally scaled coefficients using the MRPF technique. The main contributions of this work are summarized below.

- We identify the subject of computational redundancy and present methodologies which reduce implementation complexity by specifically targeting this area. Consequently, the frequency response of the given filter is not altered.
- These techniques are independent of the choice of number representation scheme.
- DCMI/ODCMI approaches are independent of the choice of coefficient word-length.
- The frame-work presented in this work can account for more general problems which consider memory overheads (by modifying edge costs), or, when given fixed resources (by solving a graph partitioning problem).

- The presented problems are mapped to well-known and extensively studied graph/set theoretic problems. Hence, efficient heuristic based solutions can be used.

In summary, this paper presents many approaches which specifically target computational redundancy reduction. One may note that there are two ways to obtain reduction in power dissipation using these approaches. First, we get a direct reduction in power dissipation due to removal of redundant computation. This advantage appears in the form of reduced switching activity [14] because of relatively fewer computational operations. Second, we can obtain multiplierless implementations, which are of immense interest in high-speed signal processing applications, and, which can also be used to further reduce power levels by employing voltage scaling [14].

This paper is organized in six sections. Section II provides a general background on FIR filtering ancl the DCM approach in [11]. Section III presents the DCMI approach for removing the computational redundancy from the filter computations. Section IV presents the optimal DCMI solution. Section V presents solutions to the hybrid problem which considers both normal as well as differential coefficients. Numerical results are presented in section VI to quantify the complexity reduction using the proposed methods. Section VII presents the MRPF technique. In section VIII, we present further numerical results. Finally, section IX concludes this paper.

II. General Background

Consider a linear time-invariant (LTI) FIR filter of length M described by an input-output relationship of the form

$$y(\mathrm{n}) = \sum_{i=0}^{M-1} c_i\, x(n - \mathrm{i}) = \sum_{i=0}^{M-1} P_i^{(n)} \tag{1}$$

In this context, c; represents the ith coefficient and $x(n - i)$ denotes the data sample at time instant n – i. $P_i^{(n)}$ represents the partial product $c_i x(n - \mathrm{i})$ for i = 0, 1, ..., M – 1 computed at time instant n. Figure 1 shows a graph G = {V, E) representation of a 4-tap (M= 4) FIR filter in which each vertex represents a coefficient and the edge $E_{i,j}$, i, $j = 0, 1, 2, 3$ represents the resources required to multiply a data sample with the preceding vertex (i.e. coefficient c,). If an array multiplier is used to compute the products, $E_{i,j}$ represents the number of rows of adders required to implement the multiplier and is given as the number of 1-bits in c_i. M parallel multipliers are required to obtain a parallel implementation of the M-tap filter. $E_{i,j}$ depends only on the number representation scheme and the type of multiplier employed. Note that G is undirected and E;,$_j = E_{j,i}$ for all i, $j = 0, 1, \ldots, M - 1$.

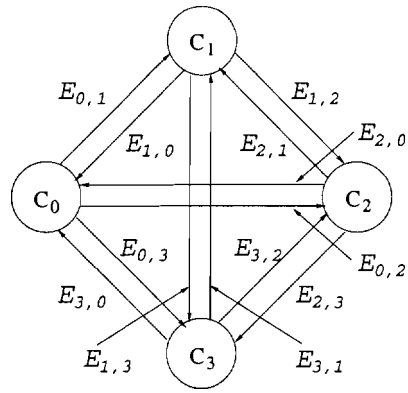

Fig. 1. Graph representation of an example filter with M = 4.

With the above interpretation of the graph, the output in equation 1 can be calculated by a tour along the graph at time instant n. Figure 2(a) shows one such tour in G which consists of edges $E_{i,(i+1)modM}$, i = 0, 1, ..., M – 1 for an $M = 8$ tap filter. The coefficients are applied such that c_{j+1} follows c_j, $j = 0, \ldots, M-2$. The appropriate data sample with the corresponding coefficient are shown next to the edges. The total resources required to compute the output given by equation 1 at time instant n is given by the sum of resources required to compute the partial products ($P_i^{(n)}$'s) along each edge in the tour. At the next time instant, $n+1$, each data sample $x(i)$, i = n, n – 1, ... n – M + 1 in the graph is replaced by $x(i+1)$. The outputs of the filter at time instants n – 1 and n are given as

$$\begin{aligned} y(n-1) &= c_0\, x(n-1) + c_1\, x(n-2) + \ldots + c_{M-1}\, x(n-M) \qquad (2)\\ &= P_0^{(n-1)} + P_1^{(n-1)} + \ldots + P_{M-1}^{(n-1)} \\ y(n) &= c_0\, x(n) + c_1\, x(n-1) + \ldots + c_{M-1}\, x(n-M+1) \qquad (3)\\ &= P_0^{(n)} + P_1^{(n)} + \ldots + P_{M-1}^{(n)} \end{aligned}$$

Next, let us view the first order *differential coefficient method* (DCM) in the context of this graph. The main idea in DCM approach is to compute coefficient differences $\Delta c_{i+1} = c_{i+1} - c_i$, i = 0, 1, ... M – 2 and using these to compute the partial products. Let $E_{i,i+1}$ represent the resources required to compute the product of $c_{i+1} - c_i$ with the corresponding data sample $x(n - i - 1)$ at time instant n for $i = 0, 1, \ldots, M-2$. Then each vertex can be replaced by the differential coefficient $c_{i+1} - c_i$ except for c_0. The partial product $P_i^{(n)}$ is computed by adding $(c_i - c_{i-1})x(n - i)$ to $P_{i-1}^{(n-1)}$. $P_i^{(n)}$ thus obtained is stored in memory for computing $P_{i+1}^{(n+1)}$ in future and removed subsequently. Hence, multiplication of c_i with x(n – i) is replaced by

addition of $P_{i-1}^{(n-1)}$ with the product $(c_i - c_{i-1})x(n - i)$. Figure 3 shows the implementation of the example filter. The overhead *add* operations are performed by using memory which stores the partial products. The cost of overhead add operations can be accounted for by adding 1 to each edge in G. Since, most filters of interest are symmetric, the later half of the filter "folds-over" as shown in figure 3. Note that the implementation of figure 3 yields $2P_i^{(n)}$, therefore, one can obtain $P_i^{(n-1)}$ from this value by a simple *right shift* operation and storing these values in a memory for use at next time sample.

The authors noticed in [11] that in shared-multiplier based implementations, this approach reduced power due to smaller word-lengths in the multiplication operation. Higher orders of differences may also be considered, however, to understand our approach for obtaining DCMI filters, it is enough to consider the first order DCM explained above.

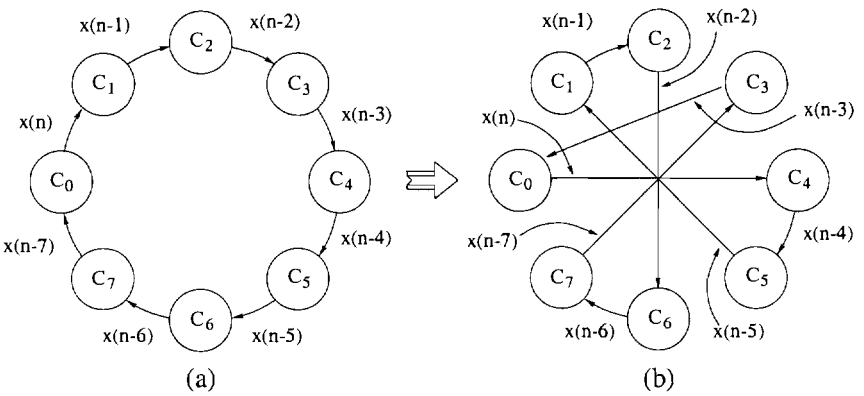

Fig. 2. Graph representation of an example filter with $M = 8$.

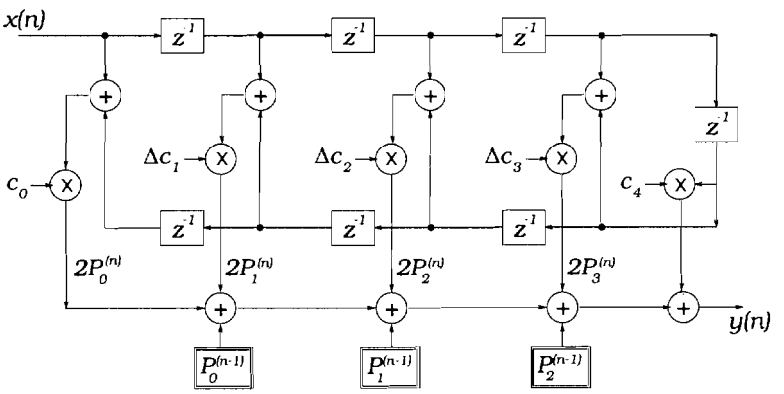

Fig. 3. Implementation of the 8-tap example filter.

III. The DCMI Approach

Consider the tour in figure 2(b). Suppose that this order yields differential coefficients which are simpler to implement than the order shown in figure 2(a) (e.g. they may be powers-of-two), and hence, the implementation so obtained has lower complexity. Note that in this example, the ordering is given by $c_0, c_4, c_5, c_1, c_2, c_6, c_7, c_3$. The corresponding data sample x(n – i) migrates from the edge $E_{i,j}$ to $E_{i,k}$, such that if T' is the new tour, $E_{i,k} \in T'$, k $\neq j$. This is shown in figure 2 where $x(n$ – i) now refers to the new edge originating at c_i. With the new ordering of figure 2(b), we get the partial products at various time instants in the order shown in table I. For simplicity in notation, let $\mathcal{K} = \{k_0, k_1, \ldots, k_{M-1}\}$ be the set representing the indices of

Time Instant	Partial Products							
$n+4$	c_0 x(n+4)	c_4 x(n)	c_5 x(n-1)	c_1 x(n+3)	c_2 x(n+2)	c_6 x(n-2)	c_7 x(n-3)	c_3 x(n+1)
$n+3$	c_0 x(n+3)	c_4 x(n-1)	c_5 x(n-2)	c_1 x(n+2)	c_2 x(n+1)	c_6 x(n-3)	c_7 x(n-4)	c_3 x(n)
$n+2$	c_0 x(n+2)	c_4 x(n-2)	c_5 x(n-3)	c_1 x(n+1)	c_2 x(n)	c_6 x(n-4)	c_7 x(n-5)	c_3 x(n-1)
$n+1$	c_0 x(n+1)	c_4 x(n-3)	c_5 x(n-4)	c_1 x(n)	c_2 x(n-1)	c_6 x(n-5)	c_7 x(n-6)	c_3 x(n-2)
n	c_0 x(n)	c_4 x(n-4)	c_5 x(n-5)	c_1 x(n-1)	c_2 x(n-2)	c_6 x(n-6)	c_7 x(n-7)	c_3 x(n-3)
$n-1$	c_0 x(n-1)	c_4 x(n-5)	c_5 x(n-6)	c_1 x(n-2)	c_2 x(n-3)	c_6 x(n-7)	c_7 x(n-8)	c_3 x(n-4)
$n-2$	c_0 x(n-2)	c_4 x(n-6)	c_5 x(n-7)	c_1 x(n-3)	c_2 x(n-4)	c_6 x(n-8)	c_7 x(n-9)	c_3 x(n-5)
$n-3$	c_0 x(n-3)	c_4 x(n-7)	c_5 x(n-8)	c_1 x(n-4)	c_2 x(n-5)	c_6 x(n-9)	c_7 x(n-10)	c_3 x(n-6)
$n-4$	c_0 x(n-4)	c_4 x(n-8)	c_5 x(n-9)	c_1 x(n-5)	c_2 x(n-6)	c_6 x(n-10)	c_7 x(n-11)	c_3 x(n-7)

TABLE I

Partial products at different time instants.

coefficients in the new ordering. Hence, for the example in figure 2(b), $\mathcal{K} = \{0, 4, 5, 1, 2, 6, 7, 3\}$. Then, the new differential coefficients for the order sequence in $\mathcal{K}$ are given by $\Delta\tilde{c}_i = c_{k_{i+1}} - c_{k_i}$, i = 0, 1, ... M – 1 and we can calculate the partial products using

$$P_{k_i}^{(n)} = (c_{k_i} - c_{k_{i-1}})x(n - k_i) + P_{k_{i-1}}^{(n-k_i+k_{i-1})} \tag{4}$$

for i = 1, ..., M – 1 (the first partial product $P_{k_0}^{(n)}$ is computed directly as $c_{k_0}x(n - k_0)$) where i = 0, 1, ... M–1. As an example, consider the computation of $P_4^{(n)} = (c_4 - c_0)x(n-4) + P_0^{(n-4)}$. Similarly, we can compute $P_4^{(n-1)} = (c_4 - c_0)x(n-5) + P_0^{(n-5)}$. Figure 4 shows the implementation details of an 8-tap filter using the DCMI approach assuming that it is asymmetric (in symmetric filter case, half of the filter "folds-over" similar to the example shown in Figure 3.). Figure 4(a) reveals the structure of the DCMI filter. Note that the implementation of this example filter requires reference to future values of partial products ($P_5^{(n+4)}$ and $P_7^{(n+4)}$). The partial products required to compute $P_i^{(n)}$'s for $i = 0, 1, \ldots, M - 1$ are shown in figure 4(b). Clearly, at any time

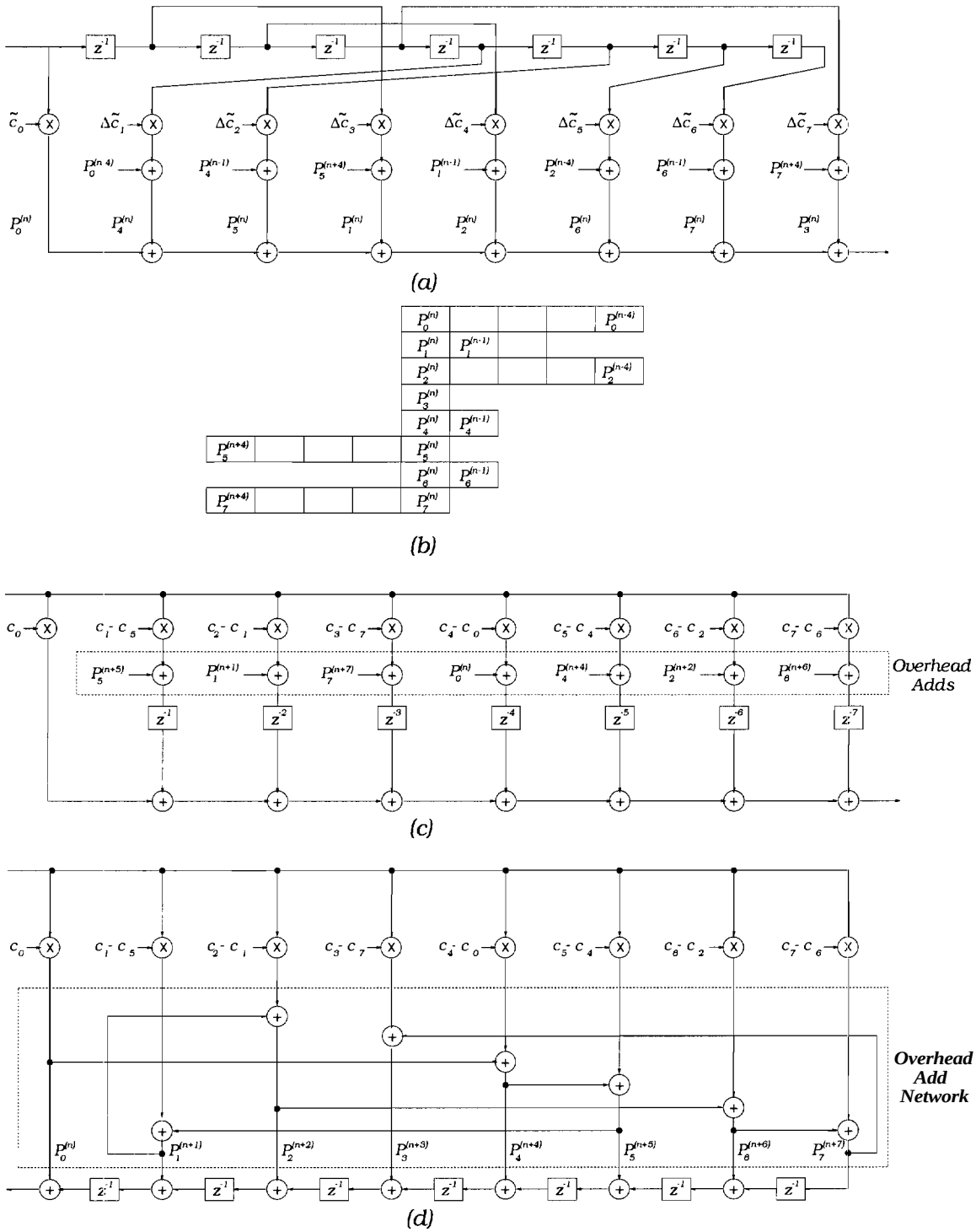

Fig. 4. Implementation of the 8-tap example filter using DCMI.

instant n, only M such products are required.

We can simplify the implementation shown in figure 4(a) by re-timing the filter. The first step

is to move the delay elements to the multiplier inputs. Consequently, the *ith* branch containing a multiplier has i delay elements on it after the first step is completed. Next, we can move the delay elements further down such that the the multiplier precedes the delay elements, and then, move them even further down such that the overhead add operations also precede these delay elements as shown in figure 4(c). The entries in table I show the partial proclucts needed to compute the output at different time instants. These entries are helpful in determining the correct partial product terms after moving the delay elements across the overhead partial product add operations. Finally, the delay elements are moved out of these branches to get the implementation shown in figure 4(d). Using the entries shown in table I we obtain the adder network for the re-timed implementation shown in figure 4(d). Hence, the filter output is available with an extra delay equal to M-adders due to the overhead add network. However, the structure of figure 4(d) can be pipelined to eliminate this delay. The overhead memory needed to store the partial product values is M for any coefficient ordering.

The DCMI approach computes the set $\mathcal{K} = \{k_0, k_1, \ldots, k_{M-1}\}$, such that the coefficient sequence $c_{k_0}, c_{k_1} \ldots, c_{k_{M-1}}$ yields the least number of resources required in the implementation. In order to compute $\mathcal{K}$, we use the graph G = (V.E) (see figure 1) in which the set V represents vertices $\{c_0, c_1, \ldots, c_{M-1}\}$ for an M-tap filter and E represents the edges, $E_{i,j}$, for $i, j = 0, 1, \ldots,$ M – 1. The edge $E_{k_j,k_{j+1}}$ connects vertex $c(k_j)$ to $c(k_{j+1})$ and represents the number of adders required to represent the difference $c_{k_{j+1}} - c_{k_j}$ in a given number representation scheme. Hence, the values assigned to the edges take into consideration the scheme used for number representation. As an example, if SM number representation is used, $c(k_i) = 17$, and, $c(k_{i+1}) = 33$, then $E_{k_j,k_{j+1}}$ is assigned a value of 1 because $c_{k_{j+1}} - c_{k_j} = 33 - 17 = 16$ requires only one adder in implementation of the multiplier. Note that G is undirected and complete [15]. There are M elements in V and M(M – 1)/2 elements in E. Hence, $|V| =$ M and $|E| =$ M(M – 1)/2 independent of the word-length or the number representation scheme used in the filter implementation.

The implementation which constructs a tour with least number of resources (total number of adders) can be obtained by computing the Hamiltonian path [15] with smallest weight in G. A Hamiltonian path is defined as a path which visits each vertex exactly once. In our work, we compute the Hamiltonian cycle instead of the Hamiltonian path. A Hamiltonian cycle is a simple cycle [15] in which each vertex in G is visited. We can remove any link in a Hamiltonian cycle to

obtain a Hamiltonian path. This offers us added convenience as we can select the first coefficient such that the first column computation ($P_0^{(j)}$, for $j = 0, 1, \ldots,$ n) requires only one adder, rather than a full multiplier. This is always possible if one of the coefficients is always fixed to a known power-of-two value and the remaining coefficients are calibrated with respect to it. For example, if c_4 in the filter in figure 2 is fixed at 2^{15} in a 16-bit SM representation scheme, and the graph in figure 2(b) represents the minimum weight Hamiltonian cycle for this filter, then the DCMI implementation would use the sequence $\{c_4, c_5, c_1, c_2, c_6, c_7, c_3, c_0\}$, thereby, avoiding the use of a full multiplier for the first partial product column computation. Hence, Hamiltonian cycle computation is more advantageous.

The Hamiltonian cycle can be solved by employing one of the known methods of solving the traveling *salesman* problem (TSP) [15], [16]. In our work. we use two well-known approaches to obtain the Hamiltonian cycle for a given graph. The first approach uses a greedy strategy which starts at a given node and extends the cycle in a depth-first search (DFS) manner. Initially, all nodes are colored white and the start node is initialized to a given node. Next, it looks at the white colored neighboring nodes of the given *start* node and selects the one which can be reached using the smallest weight edge (minimum resources). The selected node becomes the start node in the next step and is colored black. This process is repeated till all the nodes are colored black. Since the graph is complete, this method produces a tour by visiting each node exactly once. The complexity of this algorithm is $\Theta(|V| + |E|) = O(M^2)$ [15]. This algorithm is repeated by initializing the start node to each vertex in V. Hence, the complexity of the greedy approach used in this work is $\Theta(M^3)$.

The second popular approach used for solving the TSP is the heuristic algorithm due to Lin and Kernighan [16] (LK algorithm). The basic approach in this method is to complete a tour and then perforrn a local search to improve the tour. When an improvement is found, the algorithm does not necessarily use it immediately, but continues its search hoping to find an even greater improvement.

A. Second *Order DCMI*

Similar to the DCM [11], we can use higher order differential coefficients to define higher order DCMI. Let $\delta^1_{i-1,i}$ represent the first order coefficient difference, c; $- c_{i-1}$, and $\delta^2_{i-2,i}$ represent the second order coefficient difference, $(c_i - c_{i-1}) - (c_{i-1} - c_{i-2})$ = c; $- 2c_{i-1} + c_{i-2}$. Then, it can be

shown that the partial product $P_i^{(n)}$ can be calculated as [11]

$$P_i^{(n)} = c_{i-1}x(n-i) + \delta^1_{i-2,i-1}x(n-i) + \delta^2_{i-2,i}x(n-i) \tag{5}$$

where $i = 2, 3, \ldots, \mathrm{M} - 1$. Hence, using two overhead storage and two addition operations per partial procluct, we can implement the second order DCM as explained in detail in [11]. It can be verified that the second-order DCMI can be obtained by computing

$$P_{k_i}^{(n)} = c_{k_{i-1}}x(n-k_i) + \delta^1_{k_{i-2},k_{i-1}}x(n-k_i) + \delta^2_{k_{i-2},k_i}x(n-k_i) \tag{6}$$

for i $= 2, 3, \ldots, \mathrm{M} - 1$, where k_i's give the ordering sequence for the second-order DCMI. Hence, the second order DCMI requires twice as much storage and add overheads as the first order DCMI. However, similar to the first-order DCMI, by choosing c_{k_0} to be a power-.of-two, we can eliminate the full multiplication in the computation of the first column of partial products.

The second-order DCMI problem cannot be solved using the graph representation presented in section III. This is because in the second order DCMI, a second-order differential coefficient, $\delta^2_{i-2,i}$, requires reference to three coefficients, c;, c_{i-1} and c_{i-2}. Hence, if we were to use an edge to express the number of adders required to implement a multiplier with $\delta^2_{i,j}$ $(i \neq j)$ at one input, we would require counting the number of adders required to implement $\delta^2_{i,j}$ in the given number representation scheme. For a given M-tap filter, the second order differential coefficients comprising c_i and c_j as the end points would be $c_j - 2c_k + c_i$, where k $= 0, 1, \ldots,$ M--1, k $\neq i$, k $\neq j$ and, hence, it would require M $-$ 2 edges between coefficients c; and c_j in the graph. Therefore, the graph representation of section III needs to be modified to account for all possible (M $-$ 2) intermediate nodes between the given two nodes. Figure 5 shows the modified graph for a 4-tap

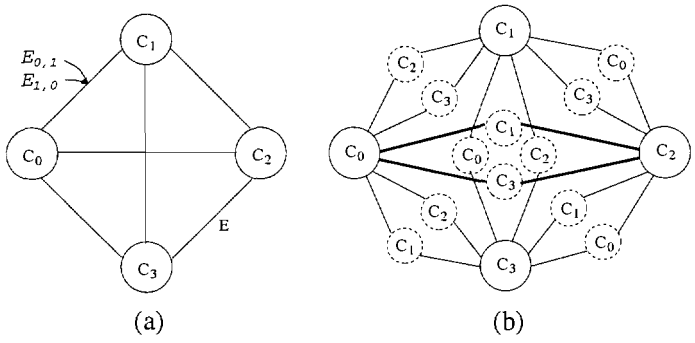

Fig. 5. Graph (G and G) Representations of an Example Filter with $M = 4$.

filter for second order DCMI problem. The vertices are represented by continuous circles. Each

pair of coefficients has $M - 2$ edges between them. This is shown using dashed circles which indicates the intermediate vertex corresponding to the edge. Note that the dashed circles do not represent vertices, rather, these illustrate the vertex considered to be the intermediate vertex in the particular edge. Hence, the modified graph, G = (V, E), for the second order DCMI can be obtained from G by inserting $M - 2$ edges between each pair of edges. In the new graph, $|V| = M$ as in G, and, $|E| = M(M-1)(M-2)/2$.

Next, we need to formulate rules for traversing G. Let the edge between vertices c; and c_j, with intermediate vertex c_k be represented as $E_{i,k,j} \in \tilde{\mathrm{E}}$, i, $j, k = 0, 1, \ldots, M - 1$, i # j # k. Now, if $E_{i,k,j}$ is traversed, this implies that we have selected the coefficient order c_i followed by c_k followed by c_j. Hence, c_i and c_k have already been visited and no subsequent edge may be visited which has c_i or c_k as an intermediate or terminal node. The only exception to this rule is when all vertices have already been visited and the tour is completed by one more step. In that case, the first node from which the tour computation was initially started must be visited as the terminal vertex.

Consider the bold path in figure 5, for example. This path shows a valid tour represented by the coefficient sequence c_0, c_1, c_2, c_3 and contains two edges $E_{0,1,2}$ and $E_{2,3,0}$. Then, after arriving at c_2, we cannot visit c_1 because it has already been visited through $E_{0,1,2}$. Further, c_0 can only be visited as the terminal node in order to complete the tour, but it cannot be used as an intermediate node. Hence, $E_{2,3,0}$ is the only edge which can be visited without violating G traversal rules. Therefore, for any $k \in 0, 1, \ldots, M - 1$, if c_k has been visited, this implies that before the next edge is traversed, all edges in the graph with c_k as the intermediate vertex must be disallowed. Similarly all edges originating from the vertex c_k must also be disallowed.

With the above rules, it is possible to devise a greedy algorithm which would start at a given initial node start and constructs a tour which visits all the vertices in the graph based on the best selection at the given time. At each step, the algorithm keeps track of three vertices, start, middle and last. This corresponds to the coefficient order $c_{start}, c_{middle}, c_{last}$. Initially, all nodes are colored white and a user selected node, initial, is taken as the start node. Next, a decision is taken at c_{start} and the best edge $E_{start,middle,last}$ is selected such that c_{middle} and c_{last} are white node:;. Next, c_{middle} is marked black and it becomes the next start node. Similarly, c_{last} becomes the new middle node and search for the next best c_{last} is performed such that the start and middle nodes are already known and c_{last} must be a white node other than initial. If no

such node can be found, then the tour is completed by selecting $c_{last} = c_{initial}$. In terms of figure 5, if the tour shown in bold were to be the best tour, the edge sequence visited will be $E_{0,1,2}, E_{1,2,3}$ and $E_{2,3,0}$ which corresponds to the coefficient order c_0, c_1, c_2 and c_3. The algorithm is repeated by initializing the *start* node to each vertex in V.

We can also use a better tour computation scheme such as the LK-algorithm. In this case, the basic LIC-algorithm must be modified such that it does not violate the graph traversal rules outlined above. The main idea is to perform a local search around a given tour to find a better tour. This is done by forming a S-path shown in figure 6. Starting at a node v, the algorithm tries improvement on both its neighboring edges. When a node w is located such that the cost of S-path shown in figure 6(b) is smaller than the cost of the initial tour T, the S-path is converted into another tour which updates the best tour found so far. Figure 6(c) shows the improved tour obtained using the S-path. We note that the modification required in this algorithm is to move the intermediate nodes p and r so that no rules are violated while improving the tour. We note that this is not the only possible approach. Better solutions may be obtained by considering the best edge $E_{u,q,w}$, where q may be any node in T other than u and w. The tour may be completed such that none of the graph traversal rules are violated on $\tilde{G}$. The approach presented in this paper is the simplest way to modify the LK-algorithm such that it can execute on $\tilde{G}$.

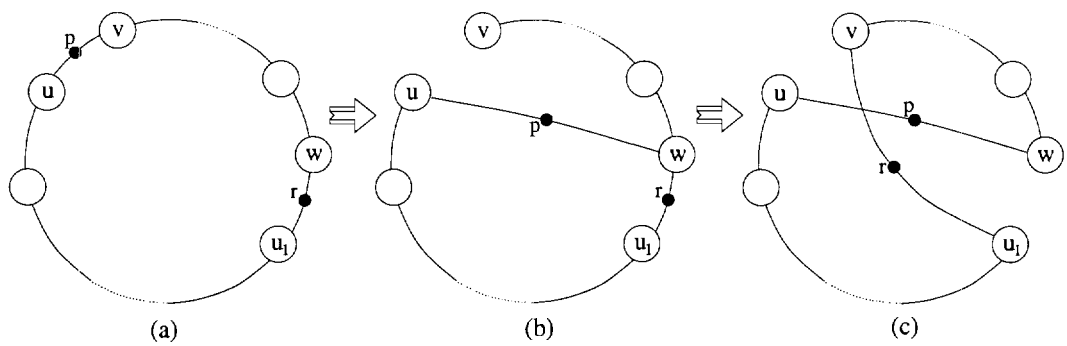

Fig. 6. Tour improvement in modified LK-algorithm.

IV. The Minimum Spanning Tree Solution

In the previous sections, we considered DCMI solution based on cycles in the graphs G and $\tilde{G}$. The primary motivation to pursue cycle based solutions was the observation that DCM can be viewed as a special case of the general framework provided in the previous sections. A major advantage of cycle based solution is the regularity of the resulting solution and implementation.

This can be advantageous in some filter implementations. For example, one may consider an application where filter exhibits some degree of adaptation so that coefficients are differenced "on-the-fly" and multiplications are performed by serial additions using a few fixed number of adders. Then, the coefficients can be stored in the order dictated by $\mathcal{K}$ and sequentially accessed from a coefficient memory of size M. Hence, cycle based solution offers an advantage of smaller overhead of coefficient storage.

In the case of a parallel implementation of filter, coefficients are pre-computed and $P_i^{(n)}$'s are obtained by adding appropriate previously computed partial products. Hence, we are not restricted to cycle based solutions, since, a differential coefficient involving a given c, can be obtained using any c, – c,, ɪ ≠ ι. With this observation, we explore the optimal solution to the DCMI problem using the *minimum* spannzng tree (MST) [15] on G. The MST of a graph G = {V, E) is defined as an acyclic subset of edges zn E which connects all of the vertices *in* V such that the sum of weights of these edges is minimized. Hence, MST on G gives a coefficient order such that all the vertices of the graph are visited once and the total resources required to implement the differential filter are minimum.

The best known algorithm to compute the MST executes in $\Theta(|E| + |V|log|V|) \approx \Theta(|V|^2)$ run time by employing Fibonacci heaps [15]. A simple algorithm by Prim runs in $\Theta(|V|^2log|V|)$ time [15]. Hence, the coefficient ordering obtained through an MST requires the least amount of add operations to compute the output. We refer to this solution as the optimal *differential coefficients* for *multiplierless* implementation (ODCMI) technique. For the problem of using least amount of resources in forming the partial products in equation 1, MST yields the optimal solution to the problem of minimizing the number of add operations given all the coefficients are differential coefficients. The major advantages of this approach is the simplicity of the algorithm and small run-time. Further, it inherits all the benefits of graph representation outlined in section I.

Figure 7 shows the implementation of an FIR filter using the MST solution (MST shown in figure 7(b)). The coefficient sequence applied using the MST of G is obtained by applying the differential coefficients $c_{child} - c_{parent}$, where c_{child} and c_{parent} pair consists of all possible parent-child pairs in the MST (leaf-nodes have no child). The parent of the root node of the MST is defined as 0. For example, the MST in figure 7(b) yields the coefficients $c_0, c_1 - c_0, c_2 - c_3, c_3 - c_0, c_4 - c_1, c_5 - c_2, c_6 - c_3, c_7 - c_3$, as shown in figure 7(a). Let $\mathcal{P} = \{p_0, p_1, \ldots, p_{M-1}\}$ and $\mathcal{Q} = \{q_0, q_1, \ldots, q_{M-1}\}$ denote the index sets of parent and child nodes, respectively. In the

above example, $\mathcal{P} = \{0, 0, 3, 0, 1, 2, 3, 3\}$ and $\mathcal{Q} = \{0, 1, 2, 3, 4, 5, 6, 7\}$. Then, the partial products can be calculated using

$$P_i^{(n)} = (c_{q_i} - c_{p_i}) x (n - q_i) + P_{p_i}^{(n-q_i+p_i)} \tag{7}$$

for i = 0, 1, ..., M – 1, i ≠ root. $P_{root}^{(n)}$ is directly calculated. As explained earlier, we ensure that at least one coefficient is set to a power-of-two, and therefore, this operation does not require full multiplication. Figure 7(c) shows the relationship of $P_{q_i}^{(n)}$'s, $i = 0, 1, \ldots, M-1$ with the previously calculated partial product values in terms of memory storage. Note that the implementation obtained is non-causal and hence output can be available after delay equal to $max(p_i - q_i, 0)$ for i = 0, 1, ..., M – 1. Figure 7(a) shows the implementation of the ODCMI filter for the MST solution shown in 7(b). The partial products required to compute $P_i^{(n)}$, $i = 0, 1, \ldots, M-1$ are shown in MST solution shown in 7(c).

As explained in section III, we can considerably simplify the implementation of 7(a) by carrying out the same re-timing steps. The resulting filter implementation is shown in 7(d). We note that the overhead memory needed to store the previous partial products is M as seen in figure 7(d). Due to the tree structure of the solution, the sequence of overhead add operations is smaller than the implementation of 4(c) where we always get a sequence of M – 1 overhead add operations. In general, the sequence is $O(logM)$ in a tree solution. Again, similar to section III, we can pipeline the series of overhead add operations which arise due to the use of differential scheme. The number of registers required to pipeline ODCMI filter is smaller than the ones required for DCMI filter.

V. Hybrid DCMI/ODCMI Schemes

The methods presented in earlier sections considered implementations which employ only differential coefficients. However, one may also consider a combination or hybrid solution which combines various orders of solutions. For example, one may consider the smallest cost tour in G which considers not only the first differential coefficients, but also the original coefficients (zeroth order coefficient) values. We will refer to this solution as a 01-hybrid solution. This approach would yield a solution which is better than the single order differential coefficients. The implementation of the 01-hybrid solution is simpler than the first order filter. The hybrid solution partitions the best solution in two sets. The first one containing coefficients which are implemented normally (i.e. directly) as multiplier inputs. The second set contains only differential

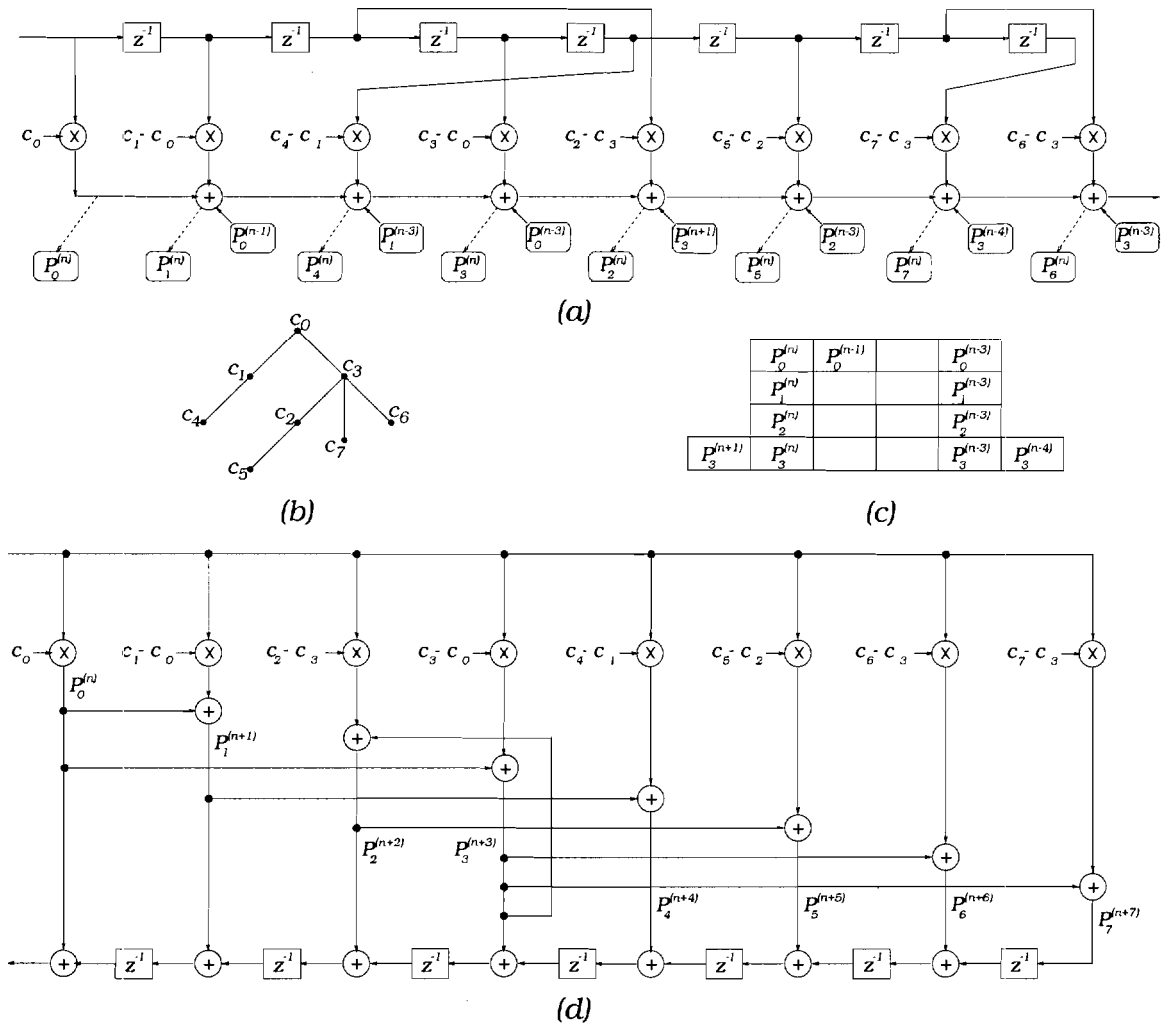

Fig. 7. Implementation of the 8-tap example filter using ODCMI.

coefficients which are implemented in the same manner as in DCMI.

The basic approach used in obtaining a 01-hybrid tour is given as follows. Given the original coefficients, we obtain the first order DCMI coefficients using the greedy or LK-algorithms. The cost of the tour is stored and for all vertices on the tour we check if the triangular inequality (Q) described in figure 8 is satisfied. This inequality checks if tour cost can be improved by removing vertices from the tour. The algorithm is given below

1. P = Original Problem with M coefficients. Set i = 0.
2. Obtain first order DCMI for P and store the tour in $T^{(i)}$ and cost of tour in *best* cost.
3. Remove all nodes satisfying condition Q from $T^{(i)}$. If no node in $T^{(i)}$ satisfies Q, then goto step 6, otherwise, set $i = i + 1$.

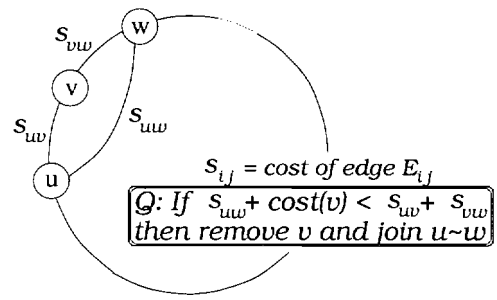

Fig. 8. Tour Improvement Using Condition T.

4. Put removed nodes in $R^{[i]}$. Call the remaining tour $P^{[i]}$.
5. If cost of $P^{[i]}$ + cost of $R^{[i]}$ < best cost, then, best cost = cost of $P^{[i]}$ + cost of $R^{[i]}$
6. 01-hybrid solution is the union of $R^{[j]}$'s for $j = 0, 1, \ldots,$ i and $T^{(i)}$. Cost = sum of $R^{[j]}$'s + cost of $T^{(i)}$.

In the above algorithm cost of $R^{[j]}$ refers to the sum of cost of each element in $R^{[j]}$, where the cost of an element in R is the number of adders required if the original coefficient value is used as one input of the multiplier. Hence, this approach removes all vertices from the tour which satisfy Q and improve the solution. The ODCMI solution is tried again on the smaller problem and the cycle is repeated till no more improvement in solution is possible.

The 01-hybrid solution for ODCMI filter is simpler to implement. The basic approach is to obtain modified graph G from G by inserting a new vertex c_M, which connects to all the other vertices c;, $i = 1, 2, \ldots,$ M – 1, such that the weight of $E_{M,i}$ is assigned a value equal to the number of adds required if the original coefficient c; were used as one input of the multiplier. Clearly, G is undirected and complete. The MST on G will be the optimal 01-hybrid ODCMI solution which requires the least number of adders to implement the multiplication. The final solution can be partitioned into two sets; the first set constituting differential coefficients in which c_M is a parent or a child, and, the second set containing the remaining pairs. The elements in the first set are the coefficients which must be applied without differencing. All pair:; in the second set are employed using the first order difference using the approach in section IV. The cost of final solution is simply the sum of edge weights on all the parent-child pairs in the MST of $\tilde{G}$.

As a final note the readers attention is diverted to the observation that the schemes for computing DCMI (ODCMI) along with all the variant solutions are significantly different from the previous attempts on designing multiplierless filters [3], [4], [5], [6], [7], [8], [9] as it uses computational reordering to reduce computational redundancy. Since, reordering does not result in

any further quantization, this approach does not compromise filter performance. Hence, these schemes yield significantly simpler and more practical and effective means of obtaining lower complexity Filter implementation.

VI. Numerical Results

We now present some numerical results to demonstrate the power and potential of the proposed approaches. Both SM and SPT number representations for implementing differential coefficients are considered. Without any loss of generality, we will assume that coefficients are normalized by the largest c_i. Two sets of results are provided to accommodate *coefficient scaling* technique which is widely used in digital filter implementation [17]. The first one uses the original coefficient values quantized to fit in a preselected *word-length*, W. The second one expresses each c_i as $c_i' \times 2^{-k}$, such that k $\geq$ 0 and $c_i' \geq 2^{W-2}$. This scheme will be referred to as *maximally scaled* coefficients. The main advantage of such scaling is that in parallel filter implementation the distortion due to quantization is minimal and this scheme yield a floating point type of filter implementation without actually using a full floating point unit. The powers of two can be trivially obtained by shifting data by appropriate amount using interconnecting wires.

Table II shows a relative comparison of filter implementations obtained using, the proposed DCMI approach when the coefficients are expressed with N = 16 bit *unscaled* numbers using SM and SPT number representations[2], respectively[3]. None of the solutions presented in this section required more than a few minutes of CPU time on a Sun Ultra 30 workstation. 'The values enclosed in parenthesis show the results for the same filters using SPT number representation. The overhead adds are excluded in this table with the purpose that the reader can focus on computation sharing obtained for the *first* and *second order* DCMI (DCMI-1 and DCMI-2, respectively) using the proposed algorithms. In addition, the M – 1 add operations in equation 1 are also excluded since these operations are identical in both norrnal as well as ODCMI implementation.

A close observation of the data in the the table reveal some interesting results. The difference in the solution:; using the greedy strategy and the LK-algorithm are negligible. Hence, near-optimal solutions are obtainable using the greedy strategy alone, as LK-algorithm is known to yield a

[2]Note that by "[Symmetric]" in the specification of filter F in the table, we mean that it is symmetric about $f = 0.5$.

[3]f_s and f_p represents normalized passband and stopband frequencies, respectively. R_p and R_s represent the passband ripple and stopband attenuation, respectively.

tour which is very close to the optimal solution [16]. The data in the table also reveals that the second-order DCMI does not offer any significant advantage as compared to the first-order DCMI. In many cases, it provides a slightly worse solution. One may conjecture that higher-order DCMI could be more useful if one were to investigate a hybrid solution which combines zeroth, first and second-order solutions. We also note that the SPT representation offers significant advantage over SM representation in many cases. Finally, we note that the number of adders per coefficient required in DCMI implementation is less than 2, in general, for SPT representation. This compares favorably to the published results of multiplierless filters in literature.

Figures 9 and 10 show a relative comparison of the average number of adders per differential coefficient obtained using the first-order DCMI solutions for SM and SPT number representations, respectively. We compare the number of adders per differential coefficient, for 8, 16 and 24 bit coefficients. The example filters considered were 28-tap PM, 41-tap LS, 119-tap PM, 172-tap LS, 131-tap PM, 170-tap LS, 151-tap PM, 217-tap LS, respectively, with specifications shown in table II. These results were obtained using the greedy strategy for first-order DCMI. We note that SPT implementations require less adders than SM implementations for all word-lengths. We also observe a linear relationship between the average number of adders per differential coefficient with the word-length. This relationship is exhibited in all the cases considered (note that the overhead adds are not included to demonstrate this point.). Further, the average number of adders per differential coefficient reduces, in general, as the length of the filter increases. We note that traditional approaches of finding multiplierless implementations for word-lengths > 16 would take enormous computational effort and may not yield good solutions. In contrast, our technique takes polynomial time, is independent of the word-length and the number representation scheme, and can be used to obtain good DCMI solutions for large filters within a few minutes of CPU time.

In the sequel, all results presented consider the overhead add network but exclude the $M-1$ add operations in equation 1 since these operations are identical in both normal as well as proposed implementations. Figures 11 and 12 show comparisons of the average number of adders per coefficient obtained using 01-hybrid ODCMI scheme for *maximally scaled* coefficients with for SM and SPT number representations, respectively. The abscissa shows the filter number used to obtain the results shown. The filters are described in table III. In both of these figures, the plot, on top shows a relative comparison of the average number of adds per coefficient. These numbers

Example Filter	M	Total Adds	DCMI-1		DCMI-2	
			Greedy	LK	Greedy	LK
A: Low-pass ($f_p = 0.25, f_s = 0.3, R_p = 3\,dB, R_s = -50\,dB$)						
Butterworth	20	94 (76)	45 (35)	44 (35)	38 (33)	38 (33)
Elliptic	6	40 (28)	19 (17)	19 (17)	13 (11)	13 (11)
PM	28	140 (118)	63 (53)	62 (53)	62 (51)	57 (49)
LS	41	206 (178)	74 (66)	72 (66)	74 (66)	70 (61)
B: Low-pass ($f_p = 0.27, f_s = 0.2875, R_p = 2\,dB, R_s = -50\,dB$)						
Butterworth	71	220 (158)	76 (68)	75 (67)	78 (64)	70 (62)
Elliptic	8	52 (40)	25 (22)	25 (22)	18 (15)	18 (15)
PM	119	578 (500)	151 (134)	145 (130)	154 (140)	153 (138)
LS	172	734 (606)	160 (146)	156 (142)	177 (165)	175 (165)
C: Low-pass ($f_p = 0.27, f_s = 0.29, R_p = 2\,dB, R_s = -100\,dB$)						
PM	189	850 (694)	183 (176)	179 (168)	198 (185)	198 (181)
LS	326	1054 (874)	202 (190)	200 (179)	228 (215)	227 (209)
D: Low-pass ($f_p = 0.25, f_s = 0.2625, R_p = 2\,dB, R_s = -73\,dB$						
PM	165	774 (598)	173 (157)	170 (155)	185 (164)	178 (159)
LS	236	912 (752)	187 (172)	185 (170)	216 (194)	212 (193)
E: Notch ($f_{p1} = 0.3, f_{s1} = 0.32, f_{s2} = 0.68, f_{p2} = .7$)						
PM	131	390 (280)	102 (74)	101 (72)	113 (96)	111 (91)
LS	170	880 (740)	202 (184)	196 (182)	217 (195)	215 (193)
F: Notch [Symmetric] ($f_{p1} = 0.2, f_{s1} = 0.22, f_{s2} = 0.38, f_{p2} = .4$)						
PM	151	428 (356)	111 (104)	110 (103)	127 (113)	126 (111)
LS	217	494 (392)	129 (111)	125 (109)	145 (126)	143 (123)

TABLE II

TOTAL NUMBER OF ADDERS IN MULTIPLIERS EXCLUDING ADD OVERHEAD ($\lfloor M/2 \rfloor$ FOR DCMI-1, $2\lfloor M/2 \rfloor$ FOR DCMI-2) OBTAINED USING DCMI FOR UNSCALED $16-bit$ SM AND **SPT** NUMBER REPRESENTATIONS, RESPECTIVELY, FOR VARIOUS EXAMPLE FILTERS.

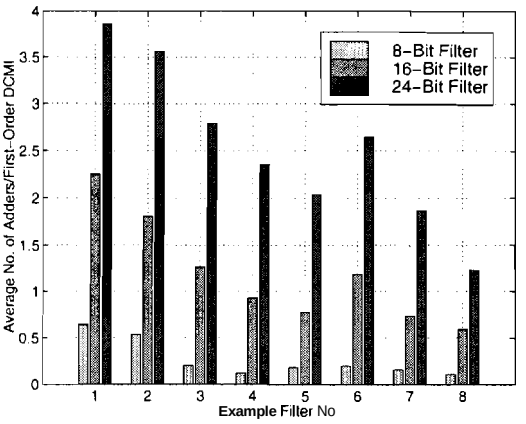

Fig. 9. Average Number of Adders per Coefficient for First-Order DCMI With Unscaled SM Number Representation. Add Overhead of $\lfloor M/2 \rfloor$ Excluded.

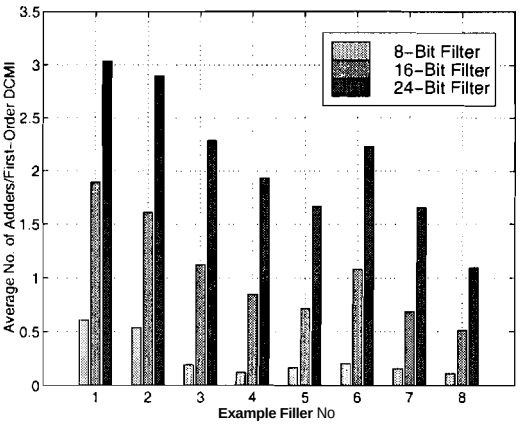

Fig. 10. Average Number of Adders per Coefficient for First-Order DCMI With Unscaled SPT Number Representation. Add Overhead of $\lfloor M/2 \rfloor$ Excluded.

are obtained by normalizing the number of add operations of the 01-hybrid ODCMI solution with the normal filter implementation. The plot on bottom shows the actual number of average add operations per coefficient for the ODCMI solution. Results are shown for $W = 8, 12, 16$ and 23 maximally scaled coefficients. Clearly, SPT requires less computational resources. Further, we observe that the relative savings is better in SM number representation as compared to SPT. This is intuitive because SPT representation reduces the required resources in the normal implementation and this is reflected as lesser gains in complexity reduction. Again, we also observe a linear relationship between the average number of adders per differential coefficient with the word-length (note that last two bars span wider range of W).

It is also evident that the 01-hybrid ODCMI filters require less than 1 adder/coefficient for

filters with M roughly ≥ 20 in the case of maximally scaled coefficients. In the case of an IIR filter implementation where maximal scaling is of paramount importance for stability reasons, the proposed solutions yield practical and viable solutions. In FIR filters, these methods yield low-complexity filters with very small quantization noise. The numerical results presented in this section also show that the first step in complexity reduction of digital filters is computational redundancy reduction rather than the conventional approach of compromising the frequency response characteristics. Significant gains can be achieved by using simple approaches as presented in this paper.

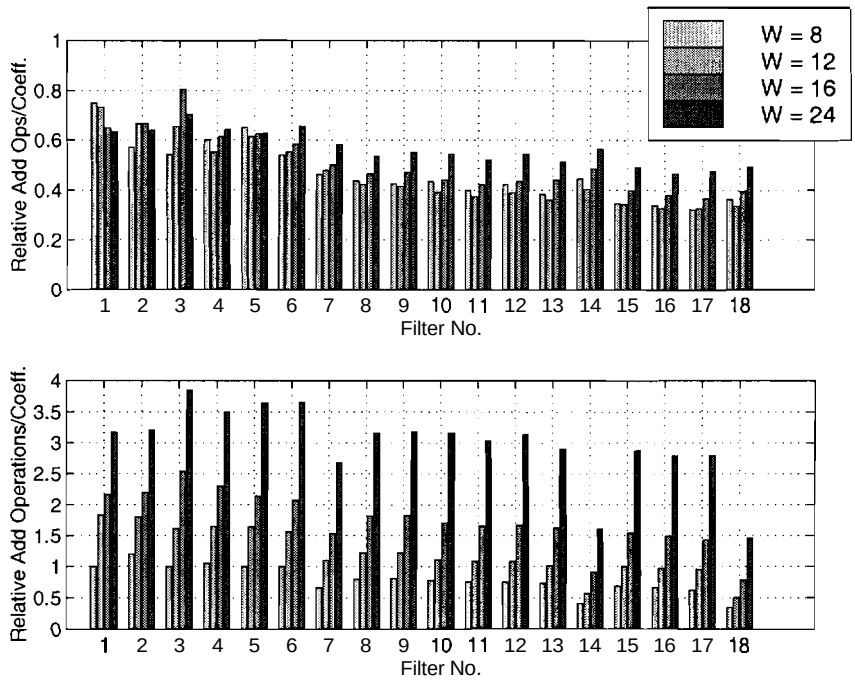

Fig. 11. Relative Comparison of the Number of Adders per Coefficient Obtained Using ODCMI for maximally scaled SM Number Representation.

VII. Minimally Redundant Parallel Filters

We now present an approach which is specific to implementations where *shifts* of computed values are available without any computational cost. In particular, this assumption is meaningful in parallel filter implementations which can trivially implement a shift operation using interconnecting wires. Alternatively, if the cost of shift operation is significantly lower than the cost of computation, the idea presented here can be easily extended to a serial SAA multiplier based implementation. To understand this approach, we consider the following example. The

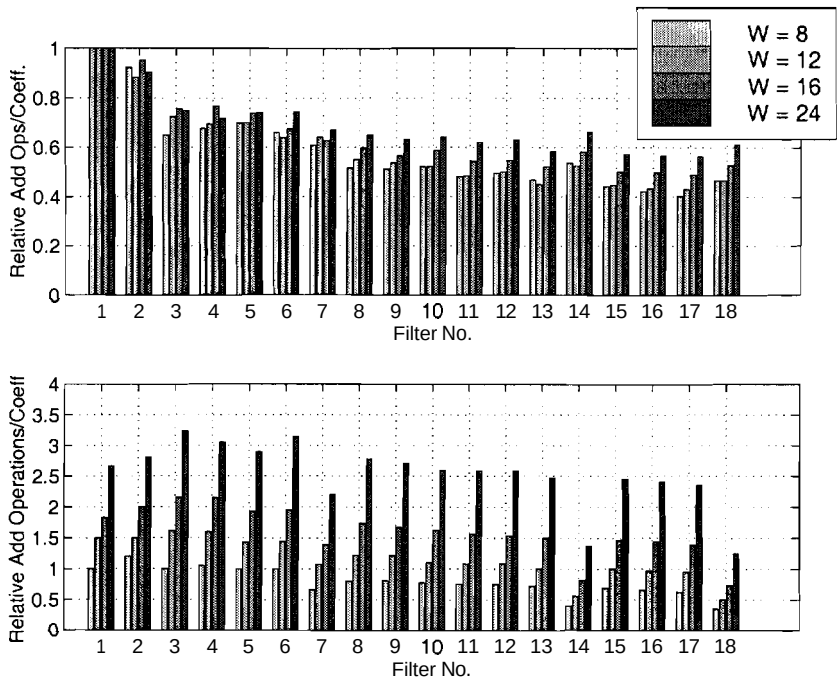

Fig. 12. Relative Comparison of the Number of Adders per Coefficient Obtained Using ODCMI for maximally scaled SPT Number Representation.

Filter No.	1	2	3	2	5	6	5	8	9	10	11	12	13	14	15	16	17	18
#Taps (M)		10	13	20	28	41	i	119	131	150	170	189	250	301				
Filter Type	EP	EP	EP	EP	BT	PM	LS	BT	PM	PM	PM	LS	PM	LS	LS	LS	PM		
	LP	LP	LP	LP	LP	LP	LP	LP	BP	BP	BP	LP	LP	BP	LP	BP	BP	BP	BP
															327	401	500	601
															LS	PM	LS	LS
															LP	BP	BP	BP

TABLE III

DESCRIPTION OF FILTERS USED IN ODCMI EXAMPLES. EP, BT, PM AND LS REPRESENT ELLIPTIC, BUTTERWORTH, PARKS-MCCLELLAN AND LEAST SQUARES FILTERS, RESPECTIVELY. BP AND LP REPRESENT BAND-PASS AND LOW-PASS FILTERS, RESPECTIVELY.

binary number 101100 can be obtained by shifting 001011 twice to the left. Hence, the product $101100 \cdot x(n)$ can be obtained free of cost from the product $001011 \cdot x(n)$, if the latter has been computed earlier. Hence, we can extend the frame-work developed in the previous sections to take advantage of this observation. Since these filters are obtained by considering, differential as well as sharing of shifted pre-computed values, the computational redundancy in these filters is significantly lower than the filters obtained using the previous approaches. We will refer to these filters as ***minimally*** *redundant parallel (MRP)* filters.

Recall the ODCMI scheme presented in section IV. A close observation of figure 7 reveals that computational redundancy in the filter shown can be further reduced if computation can be shared amongst the products, $(\Delta c_i)_{MST}x(n) = \{c_{q_i} - c_{p_i}\}x(n)$, $i = 0, 1, 2, \ldots, \mathrm{M}-1$, q_i's and pi's refer to the parent and child nodes of the MST. Without loss of generality, we will assume that the filter coefficients have been maximally scaled such that $c_i \geq 2^{M-2}$ for each $i = 0, 1, \ldots, \mathrm{M}-1$. We will let W represent the coefficient word-length. Then, the generalized differential coefficients are given as $(\Delta c_i)_{L,MRPF} = c_{q_i} - 2^{-L}c_{p_i}$, where $0 \leq \mathrm{L} \leq \mathrm{W}$ and $i = 0, 1, \ldots, \mathrm{M}-1$. Note that maximal scaling results in maximum number of distinct values of $(\Delta c_i)_{L,MRPF}$ for $0 \leq \mathrm{L} \leq \mathrm{W}$. Next, suppose that we were to implement a solution in figure 7 where each $(\Delta c_i)_{MST}$ is replaced by some $(\Delta c_i)_{L_i,MRPF}$ for $i = 0, 1, \ldots, \mathrm{M}-1$, where $0 \leq L_i \leq \mathrm{W}$. Then, $P_i^{(n)} = (\Delta c_i)_{L_i,MRPF}x(n) + 2^{-L_i}c_{p_i}x(n)$, and hence, a correction term of $2^{-L_i}c_{p_i}x(n)$ needs to be added to obtain the correct value of the partial product. We note that this term is readily available from $c_{p_i}x(n)$ by a shift operation of L_i bits. Consequently, we can obtain $P_i^{(n)}$ through any one of $(\Delta c_i)_{L,MRPF}$, $i = 0, 1, \ldots, \mathrm{M}-1$, $\mathrm{L} = 0, 1, \ldots, \mathrm{W}$, if the cost of shift operation is negligible in terms of computation.

The above is represented in the modified graph of figure 13 for the 4-tap example filter. This graph is directed, and there are $\mathrm{W}+1$ edges directed from c; to c_j for all i, j and representing the difference $c_j - 2^{-L}c_j$, $\mathrm{L} = 0, 1, \ldots, \mathrm{W}$. We will let $D_{i,j}^k$ denote the edge representing the differential coefficient $c_j - 2^{-L}c_j$ as shown in figure 13. Consequently, each vertex has $(W+1)(\mathrm{M}-1)$ incident edges. In contrast to the graphs obtained for the problems presented in previous sections, the edges in the graph of figure 13 represent the actual value of of $(\Delta c_i)_{L,MRPF}$, and not the number of resources required to obtain a product with this generalized differential coefficient. We let the value of the $(\Delta c_i)_{L,MRPF}$ represent the color of the corresponding Edge. Hence, each vertex has a maximum of $\mathrm{W}+1$ distinct colored edges entering it from another vertex and there are a total of $(\mathrm{W}+1)(\mathrm{M}-1)$ distinct colored edges in the graph.

Let each vertex represent a set of incident colored edges. Then, we can obtain M such sets for an M-tap filter. In order to implement the filter, we need to visit each vertex once only. Hence, the solution requiring least amount of resources to implement this filter comprises of a set of colors whose edges visit each vertex at least once, such that the cost of implementation of these colors is minimum. Further, if a particular color edge is selected, all edges of the same color can be obtained free of computational cost for all other vertices visited by these edges. We can

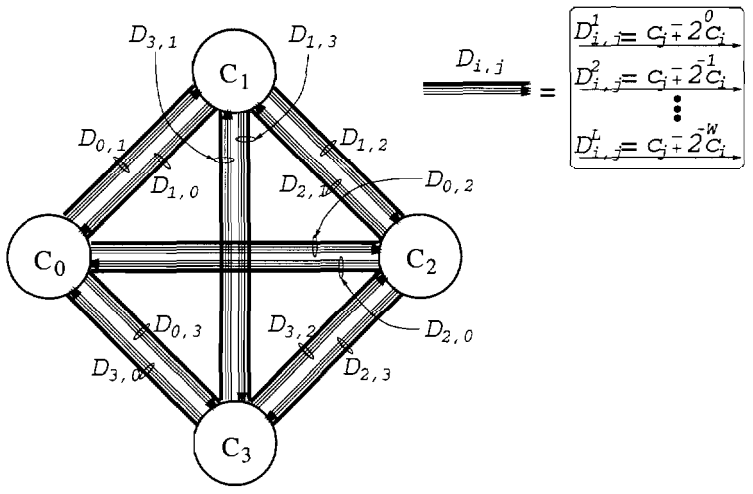

Fig. 13. Modified graph for MRP filter.

also cast this problem to an alternative equivalent problem in which all possible edge colors for a particular set of scaled coefficients define a color set. The elements of a given color set comprises the vertices visited by the edges of the respective color as shown in figure 14. The total number of vertices which are visited by *color;* are stored in *count,* field. Each color requires a particular cost of implementation which is the amount of resources required to form a product of a data value by the value represented by the color. This cost, for a given number representation scheme is stored with the color. As explained above, if *color,* is selected in the final solution, the product *color,* $. x(n)$ (equal to $\Delta c_{iL,MRPF} \cdot x(n)$ for some i and L) can be shared using interconnecting wires to obtain any other vertex in the set free of computational cost. Then, our goal is to find least cost set of colors such that the edges in these sets cover all of the vertices in the graph. This is a well known NP-complete problem called *weighted minimum set cover.* Hence, it can be solved using a good heuristic approach.

A. Solution for the MRP Filter

A greedy algorithm employed to solve the above minimum set cover problem is outlined below. It takes scaled input coefficients and computes a greedy solution for the MRP filter.

1. (Pre-process) Remove all vertices $c_j = 2^{-l}c_i$, for some l such that $j \# i$.
   Repeat for all j such that $0 \leq j \leq M-1$.
2. Construct *Vertex Sets* for all vertices (coefficients) in the modified graph.
   Vertex Set; is defined as the set of all incident edges on *c;*.

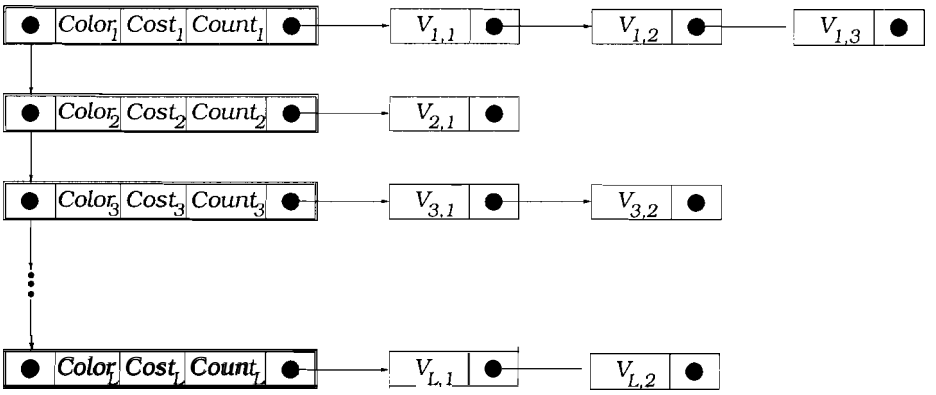

Fig. 14. The data structure for color sets.

3. Construct Color Sets from Vertex Sets. Color-Set; is defined as the set of all vertices which can be visited by the color of Color-Set;.

4. Compute the minimum cost weighted set cover problem.

(a) Initialize *MSC-Solution* to empty set.

(b) while (Color Sets are not empty) do

i. Choose and the lowest cost color which visits the maximum number of vertices. Add it to the set MSC-Solution.

ii. Recnove all vertices visited by the chosen color from the Color Sets.

(c) end *while*

(d) Greedy minimum set cover solution is the set of chosen colors in MSC-Solution

5. If any $c_i = 2^{-k} color_j$, for $i = 0, 1, \ldots, \mathrm{M} - 1$ and $color_j \in$ MSC-Solution, remove the overhead ADD operation for that coefficient. For all others, add overhead cost of **1.**

6. Add cost of colors in MSC-Solution to obtain the total implementation cost.

In the first step, all vertices which can be obtained by simple shifts of other vertices are removed from the problem. Only one of these vertices is kept (i.e. c_i). There is no computation or overhead add operation required to implement partial products involving the rernoved vertices. Hence, c_j's are obtained with cost 0. We note that. since the removed edges are $c_j = 2^{-l} c_i$, the incident colors on c_j's would have been identical to the incident colors on c;, had we chosen not to remove these vertices. Hence, the final solution will not be affected, as the choice of colors is not increased if c_j's are not removed, and visiting c_i automatically visits each c_j. It follows that step **1** does not alter the optirnality of the solution obtained for the smaller problem.

Steps 2 and **3** compute the *Vertex* and *Color Sets* as explained earlier. In step 4, any heuristic approach may be used to compute the minimum cost weighted set cover solution. In our work, we use a simple greedy approach which selects the most likely color as the one which visits most vertices and has the smallest cost. Finally, we need to account for the overhead addl operations for the differential coefficients and step *5* removes all the overhead add operations for the coefficients which can be directly be obtained from the colors selected in the solution color sets. Finally, the total implementation cost is computed for the MRP filter.

VIII. Further Numerical Results

Figures 15 – 18 show a comparison of MRPF filter complexity with the normal filter implementation for SM and SPT number representations, respectively. These results were obtained using the greedy solution presented in the previous section. The example filters used in obtaining these results are clescribed in table III. The results clearly indicate that MRP filters have lower complexity as compared to ODCMI filters. We also note that the relative savings in average number of adds required to implement the given filter are greater for SM representation. Again, this is because the average number of adds per coefficient are smaller for the normal implementation with SPT representation.

Figures 15 and 17 show that MRP filters decrease the filter complexity roughly by a factor of 2 for most cases. Of particular interest is the observation that MRPF approach is also useful in reducing the complexity of small filters. For long filters, complexity reduction is almost by a factor greater than **3.** These savings result from *reuse* of shifts of computation. Figures 16 and 18 show that for long filters, the average number of adders required per multiplication is less than 0.2 for 8-bit maximally scaled coefficients. Hence, a fully parallel implementation of an 8-bit maximally scaled 200 tap filter requires 20–40 adders to implement the full multiplication network. Consequently, the complexity of the filter is dominated by the $M - 1$ add operations in the sum of equation 1 rather than multiplications. We observe that this dramatic reduction in filter complexity does not compromise the filter transfer function response and is obtained using a simple polynomial run-time algorithm. Hence, MRPF technique offers a powerful methodology to obtain very low-complexity and fast digital filters for applications requiring very high-performance and/or low power.

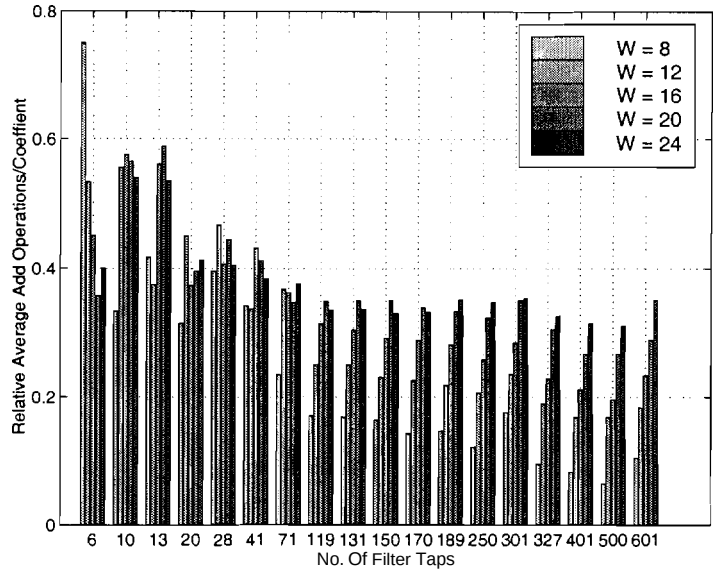

Fig. 15. Relative Average Number of Adders per Coefficient in MRP filters Using maximally scaled SM Number Representation.

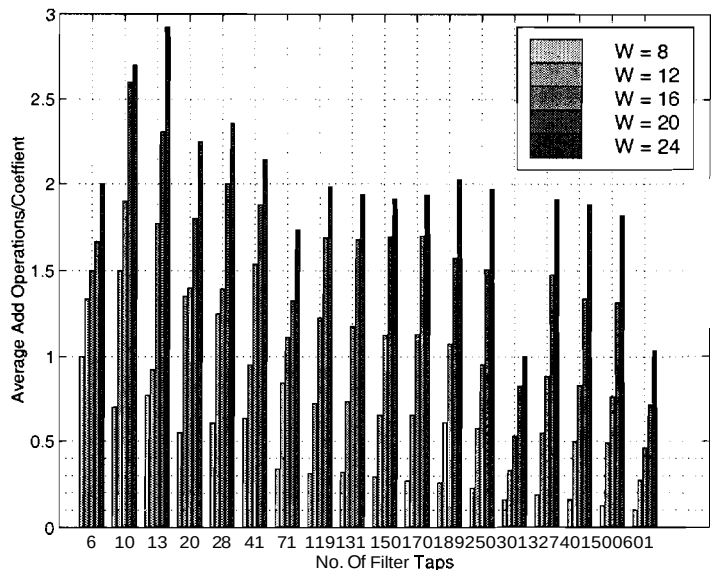

Fig. 16. Average Number of Adders per Coefficient in MRP filters Using maximally scaled SM Number Representation.

IX. Conclusion

We presented computation reduction techniques which can be used to obtain multiplierless implementations of FIR digital filters. The ideas presented in this paper are also applicable to IIR digital filters. We addressed the problem of complexity reduction for high-speed and low power

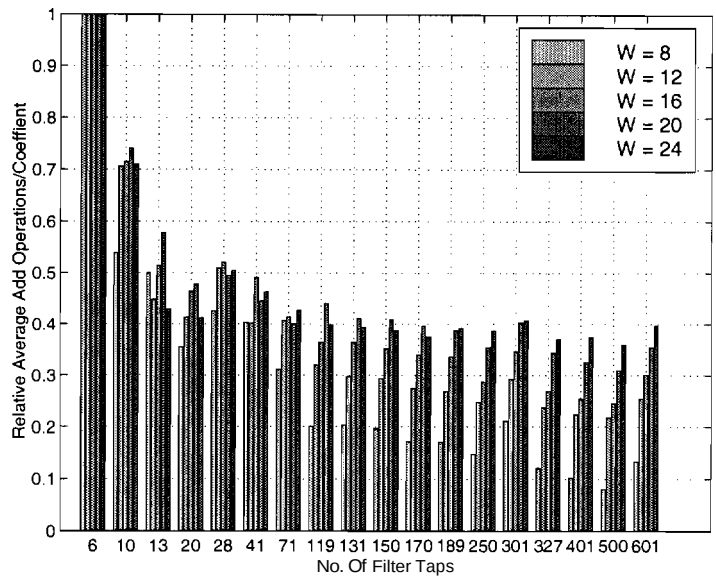

Fig. 17. Relative Average Number of Adders per Coefficient in MRP filters Using maximally scaled SPT Number Representation.

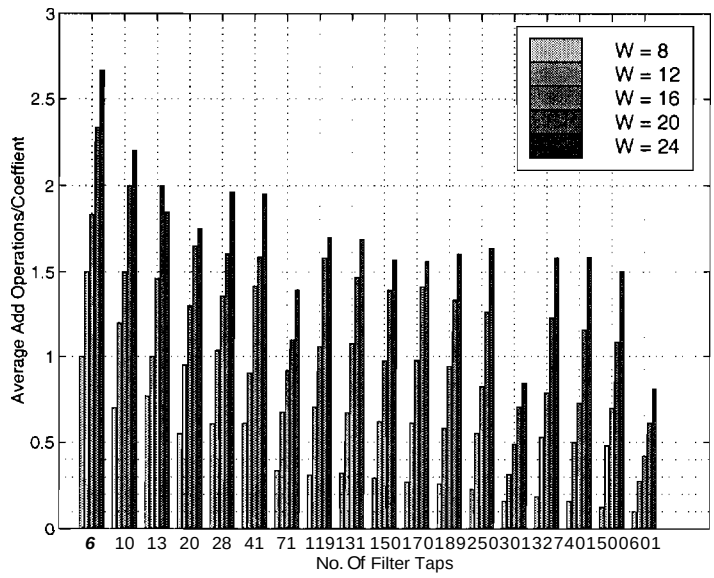

Fig. 18. Average Number of Adders per Coefficient in MRP filters Using maximally scaled SPT Number Representation.

applications by proposing systematic methodologies for reducing *computational redundancy* using computation *reordering* and *sharing*. Various approaches were presented which consider *normal*, *differential* and *hybrid* coefficients. The reordering problem was formulated using a graph in which vertices represent the coefficients and edges represent resources required in a computa-

tion involving the coefficient. Various schemes were presented which reduce filter complexity by specifically targeting computational redundancy inherent in normal filter implementations. Simple polynomial run time algorithms are presented and their power and potential was demonstrated by presenting results for large filters *(lengths* up to *600)* which showed significant gains in the number of add operations per coefficient. We also considered filter implementations in which *shifted* values of computations can be obtained using simple interconnects without incurring extra computation. We presented a methodology using which such computation can be *re-used* in other computations and showed such operations significantly reduce further computational redundancy, thereby yielding extremely simple filters. One major advantage of the proposed schemes is it that the frequency response of the desired filter is unaltered. It was shown that as low as 0.1 adders per filter coefficient are required to implement the multipliers in such filters. Hence, such filters can be used in very high-speed applications. Alternatively, using voltage scaling, one can significantly reduce the power consumption of such filters for any desired performance.

REFERENCES

[1] L. L. Scharf, "Statistical Signal Processing: Detection, Estimation, and Time Series Analysis," Addison Wesley, 1991.

[2] S. Haykin, "Adaptive Filter Theory," Prentice Hall, NJ, 1996.

[3] R. Jain, P. T. Yang and T. Yoshino, "FIRGEN: A computer-aided design system for high performance FIR filter integrated circuits," *IEEE Transactions on Signal Processing,* Vol. 39, No. 7, pp. 1655--1668, Jul. 1991.

[4] Y. C. Lim and S. R. Parker, "FIR filter design over a discrete powers-of-two coefficient space," *IEEE Trans. Acoust., Speech & Signal Processing,* Vol. ASSP-31, No 3, pp. 583-591, Jun. 1983.

[5] D. Li. and Y. C. Lim, "Multiplierless realization of adaptive filters by nonuniform quantization of input signal," *1994 IEEE International Symposium on Circuits and Systems,* Vol. 2, pp. 457-459, 1994.

[6] B. R. Horng, H. Samueli and A. N. Wilson, "The design of low-complexity in linear-phase FIR filter banks using powers-of-two coefficients with an application to subband image coding," *IEEE Trans. Circuits Syst. Video Technology,* Vol. 1, No. 4, pp. 318-324, Dec. 1991.

[7] B. R. Horng, H. Samueli and A. N. Wilson, Jr., "The design of two-channel lattice structure perfect-reconstruction filter banks using powel-s-of-two coefficients," *IEEE Trans. Circuits and Systems-I:Fundamental Theory and Applications,* Vol. 40, No. 7, pp. 497-499, July 1993.

[8] H. Samueli. "An improved search algorithm for the design of multiplierless FIR filters with powers-of-two coefficient:;." *IEEE Trans. Circuits and Systems,* Vol. 36, No. 7, pp. 1044-1047, July 1989.

[9] M. Yagyu, A. Nishihara and N. Fujii, "Fast FIR Digital Filter Structures Using Minimal Number of Adders and its Application to Filter Design," *IEICE Trans. Fundamentals,* Vol. E79-A, No. 8, pp. 1120–1128, Aug.

1996.

[10] D. A. Parker and K. K. Parhi, "Low-area/power parallel FIR digital filter implementations," *Journal of VLSI Signal Processing*, Vol. 17, No. 1, Sept. 1997.

[11] N. Sankarayya, K. Roy, and D. Bhattacharya, "Algorithms for low power and high speed FIR filter realization using differential coefficients," *IEEE Trans. Circuits and Systems*, Vol. 44, No. 6, pp. 488-497, Jun. 1997.

[12] K. Muhammad and K. Roy, "Low Power Digital Filters Based On Constrained Least Squares Solution," *In Proc. 31st Asilomar Conference On Signals, Systems, & Computers*, Nov. 2–5, 1997.

[13] M. Mehendale, S. B. Roy, S. D. Sherlekar and G. Venkatesh, "Coefficient transformations for area-efficient implementation of multiplier-less FIR filters," *Proceedings Eleventh International Conference on VLSI Design*, pp. 110-115, 1997.

[14] J. M. Rabaey, "Digital Integrated Circuits: A Design Perspective," Prentice Hall, New Jersey, 1996.

[15] T. H. Cormen, C. E. Leiserson and R. L. Rivest, "Introduction to Algorithms," The MIT Press, 1990.

[16] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank and A. Schrijver, "Combinatorial Optimization," John Wiley & Sons, Inc., 1998.

[17] E. Cooper, "Minimizing Quantization Effects Using the TMS320 Disgital Signal Processor Family," *Application Report, http://www.ti.com/sc/docs/psheets/abstract/apps/spra035.htm*, Texas Instruments, 1994.